멀티프로세서용 임베디드 시스템을 위한 UML 기반 소프트웨어 모델의 분할 기법

김 종 필[†] · 홍 장 의[††]

요 약

임베디드 시스템의 하드웨어 구성요소들에 대한 성능 고도화가 요구됨에 따라 이에 탑재될 소프트웨어의 개발 방법도 영향을 받고 있다. 특히 MPSoC와 같은 고가의 하드웨어 아키텍처에서는 효율적인 자원의 사용 및 성능의 향상을 위해 소프트웨어 측면에서의 고려가 필수적으로 요구된다. 따라서 본 연구에서는 임베디드 소프트웨어 개발과정에서 멀티프로세서 기반의 하드웨어 아키텍처를 고려하는 소프트웨어 태스크의 분할기법을 제시한다. 제시하는 기법은 UML 기반의 소프트웨어 모델을 CBCFG (Constraints-Based Control Flow Graph)로 변환하고, 이를 병렬성과 데이터 의존성을 고려한 소프트웨어 컴포넌트로 분할하는 기법이다. 이러한 기법은 임베디드 소프트웨어의 플랫폼 의존적인 모델 개발과 태스크 성능 예측 등을 위한 자료로 활용할 수 있다.

키워드 : 임베디드 소프트웨어, UML 다이어그램, 제약기반 제어흐름그래프(CBCFG), 모델 분할

A Partition Technique of UML-based Software Models for Multi-Processor Embedded Systems

Jong-Phil Kim[†] · Jang-Eui Hong[††]

ABSTRACT

In company with the demand of powerful processing units for embedded systems, the method to develop embedded software is also required to support the demand in new approach. In order to improve the resource utilization and system performance, software modeling techniques have to consider the features of hardware architecture. This paper proposes a partitioning technique of UML-based software models, which focus the generation of the allocatable software components into multiprocessor architecture. Our partitioning technique, at first, transforms UML models to CBCFGs(Constraint-Based Control Flow Graphs), and then slices the CBCFGs with consideration of parallelism and data dependency. We believe that our proposition gives practical applicability in the areas of platform specific modeling and performance estimation in model-driven embedded software development.

Key Words : Embedded Software, UML Diagram, CBCFG, Model Partitioning

1. 서 론

사회의 고도화와 전문화, 그리고 복잡성의 증대로 인하여 사회 활동을 지원하기 위한 장치나 기기들이 더욱 복잡해지고, 이러한 장치에 탑재되는 임베디드 소프트웨어의 기능 및 성능도 더욱 복잡해지고 있는 추세이다. 따라서 향후의 임베디드 소프트웨어들은 고유의 기능 수행이외에도 다양한 부가 서비스에 대한 요구사항을 충족시켜야 하며, 이로 인하여 고성능의 처리를 필요로 하고 있다[1].

따라서 본 논문에서는 고성능의 멀티프로세서 플랫폼을 기반으로 하는 임베디드 소프트웨어의 개발 기법에 대하여 연구하였다. 특히 최근에 객체지향 개념을 기반으로 하는 임베디드 소프트웨어의 개발 추세가 확산[2]됨에 따라 UML을 이용한 모델 기반의 임베디드 소프트웨어 개발에서 멀티프로세서 아키텍처를 어떻게 지원할 것인가에 대하여 고찰하였다.

임베디드 소프트웨어의 성공적인 개발을 위해서는 설계 과정에서 반드시 소프트웨어가 탑재될 하드웨어의 플랫폼이 고려되어야 한다. 특히 이기종의 다수 프로세서 - 예를 들면, ARM core, DSP, Device controller 등 - 를 갖는 하드웨어 아키텍처를 채택하는 경우에 있어서는 소프트웨어 모듈들이 어떻게 하드웨어 컴포넌트에 배치되는가가 시스템의 성능 및 자원의 효율성 측면에서 매우 중요하다. 특히 설계 및 개발에 많은 비용을 요구하는 고성능의 하드웨어 플랫폼의 경우, 소프트웨어의 부적절한 설계로 인하여 하드웨어

※ 본 논문은 2006학년도 충북대학교 학술연구지원 사업의 연구비지원에 의해 연구되었음.
† 준 회 원 : 충북대학교 전자계산학과 석사과정
†† 종신회원 : 충북대학교 컴퓨터공학 조교수(교신저자)
논문접수 : 2007년 7월 27일, 심사완료 : 2007년 9월 13일

DOI: 10.3745/KIPSTD.2008.15-D.1.87

플랫폼을 변경해야 해야 하는 심각한 문제 - 재작업을 비롯한 비용 손실 - 를 유발할 수 있다[3, 4].

따라서 본 연구에서는 UML 2.0[5]을 근간으로 하는 모델 기반의 임베디드 시스템의 개발 과정에서 소프트웨어 모델을 체계적으로 분할하기 위한 기법을 제안한다. 소프트웨어 모델의 분할은 멀티프로세서를 갖는 하드웨어 아키텍처에 적절한 작업의 분산을 위해 요구된다. 제안하는 기법에서는 요구사항을 중심으로 개발된 UML 모델을 입력으로 행위 중심의 제어 흐름 그래프(CFG, Control Glow Graph)를 생성한다. 생성한 그래프는 멀티프로세서 아키텍처의 구성요소들로 매핑하기 위해 분할되어진다. 이러한 과정을 통해 얻어진 소프트웨어 모델 분할은 MPSoC (Multi-Processor System-on-Chip)[3]와 같은 멀티프로세서용 하드웨어 아키텍처에 소프트웨어를 어떻게 배치할 것인가 하는 문제의 해결책으로 사용될 수 있다.

기존의 임베디드 소프트웨어 개발 방법에서 설계 모델로부터 태스크를 도출하여 하드웨어 요소들로 할당하기 위한 방법들[6, 7, 8]이 연구되어 왔는데, 이들은 동일한 구조를 갖는 프로세서들로 하드웨어 플랫폼이 구성되어 있음을 가정하거나, 소스 코드로부터 태스크 도출하는 기법들이다. 본 연구에서는 이와 달리 서로 다른 이기종의 하드웨어 구성요소들로 플랫폼이 구성될 수 있으며, 이러한 하드웨어 아키텍처 특성을 어떻게 소프트웨어 모델 분할에 활용할 것인가에 주안점을 두고 있다. 이와 같은 연구는 임베디드 소프트웨어의 모델링 단계에서 플랫폼을 고려한 소프트웨어의 기능 및 성능에 대한 검증 및 시뮬레이션을 가능하도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련된 주요 연구들에 대한 조사 분석하고, 3장에서는 UML 기반의 임베디드 소프트웨어 모델링 방법에 대하여 간략히 설명한다. 4장에서는 UML 모델로부터 생성되는 제어흐름 그래프를 정의하고, 5장에서는 그래프의 분할 기법에 대하여 제시한다. 6장에서는 예제 시스템을 통한 기법의 적용 및 분석 결과를 기술하고, 결론을 맺는다.

2. 관련 연구

본 연구는 UML 2.0 기반의 임베디드 소프트웨어 개발에서 플랫폼 의존적인 모델을 개발하기 위한 소프트웨어 모델의 분할에 대한 연구로써, UML 모델로부터 CFG를 생성하기 위한 방법과 생성된 CFG를 분할하기 위한 기법으로 구성된다. 이에 관한 기존 연구들을 살펴보면 다음과 같다.

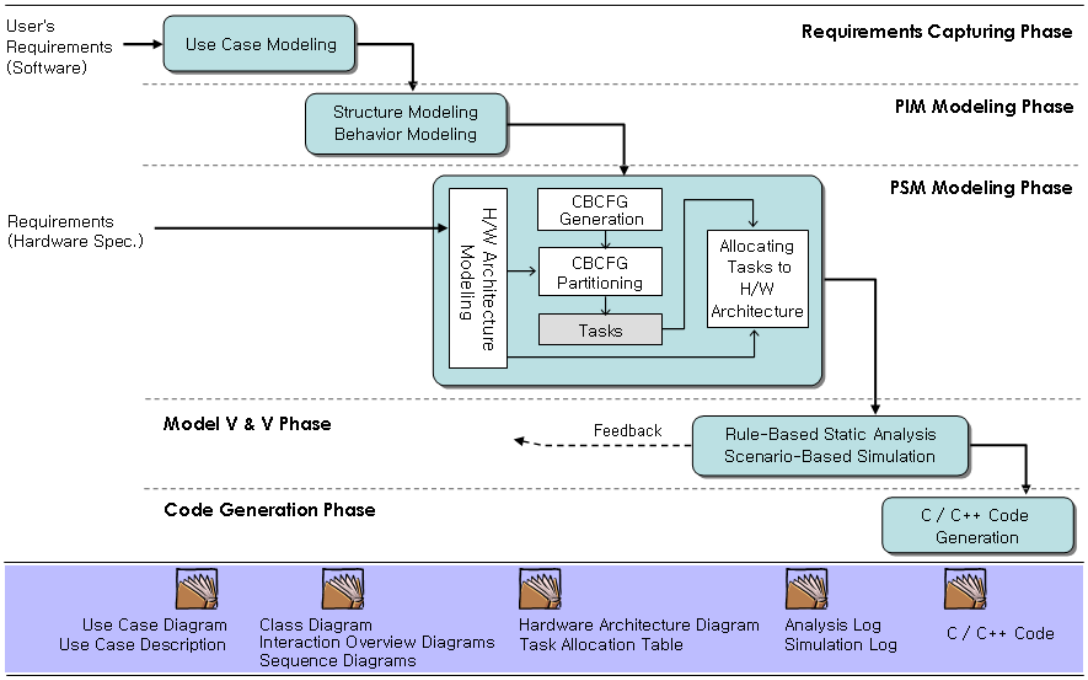
UML 모델로부터 CFG를 생성하는 대표적인 연구는 Garousi [9]에 의해 수행되었다. Garousi는 UML 2.0의 Activity Diagram (AD)을 확장하여 Concurrent CFG(CCFG)를 정의하고, Sequence Diagram(SD)을 구성하는 요소들을 CCFG 요소들로 매핑하기 위한 규칙을 제안하였다. 제안된 14개의 규칙에 대하여 OCL 언어를 이용하여 표현하고, 이들에 대하여 UML 메타 모델을 확장하였다. 그러나 이 연구에서는 단지 SD를 이용한 CCFG를 생성하는 방법을 제시하기 때문에 전체 소프트웨어에 대한 제어 흐름을 파악하기 어렵다는 문제점이 있다. 따라서 소프트웨어 모델 전체에 대한 흐름을 파악하기 위해 SD 이외에도 Interaction Overview Diagram(IOD)과 같은 모델의 제어 흐름이 반영되어야 한다. 또 다른 연구로써, Storrle[10]는 UML 2.0의 AD에 대한 제어 흐름을 페트리 넷 시맨틱으로 정의하였다. 따라서 페트리 넷의 분석 기법을 통해 AD가 갖는 행위의 속성을 분석할 수 있으나, 예외처리나 SubActivity 등에 대한 시맨틱을 정의하지 못하고 추후 연구내용으로 남겼다. 이외에도 CFG와 관련된 연구는 Honsell-Mitchell Functional Object-Oriented Calculus에 의해 CFG를 표현하여 동시성을 갖는 객체들의 제어 흐름을 분석한 Blasio의 연구[11]가 있으며, Zhao[12]는 관점지향 프로그램에 대한 CFG를 생성하여 프로그램의 이해성과 유지보수성을 향상시키려는 연구를 수행하였다. 그러나 이러한 연구들은 CFG를 생성을 위한 입력이 하나의 모델만을 취한다는 점이 본 연구와의 근본적인 차별점이라고 할 수 있다.

모델의 분할에 대한 연구는 설계 자동화 분야에서 많이 수행되었는데, 대표적으로 Hollstein에 의해 제시된 HiPART 기법[13]이다. 이는 임베디드 시스템 통합설계과정에서 하드웨어와 소프트웨어 부분을 분리하는 알고리즘으로써, VHDL (VHSIC Hardware Description Language) 언어로 기술된 명세를 CDFG(Control Data Flow Graph)로 변환하고, 노드간에 존재하는 통신량 및 데이터 공유정도를 고려하여 그래프를 분할한다. 그러나 이 연구는 단지 하드웨어 구조를 표현한 그래프 분할에 주안점을 두었다. 다른 연구는 C/C++ 분석을 통해 OCAPI-xl 모델을 생성하고, 이를 분할하여 병렬성을 분석하는 기법으로 Vanmeerbeek[14]에 의해 제시되었다. OCAPI-xl은 시스템이 갖는 메시지 통신, 세마포어, 공유 변수 등의 분석을 통해 프로세스간의 병렬성을 도출하기 위해 사용하는 C++ 클래스 라이브러리의 일종이다. 이 방법은 소스코드를 기반으로 하고 있기 때문에 모델 수준에서의 분할에 적용하기 어렵다. 또한 바이오 정보처리를 위한 병렬 컴퓨터 기반의 모델 분할 연구가 Cornelis[15]에 의해 진행되었는데, 이는 뉴럴 네트워크를 구성하는 돌기 세포들을 일정한 비율로 분할하여 프로세서의 부하(workload)를 균등하게 할당하기 위한 기법이다. 이 기법에서는 모든 프로세서가 동일한 구조를 가지며, 분할 기법도 단순히 균등한 로드 밸런싱을 위해 이루어진다. 기타 관련 연구로써는 Hering[16], Li [17]등에 의해 진행되었으며, 이러한 연구들도 하드웨어 구조를 분할하는 것에 주안점을 두고 있어서, 소프트웨어 모델의 분할과는 차이점을 보이고 있다.

3. 임베디드 소프트웨어 모델링

3.1 모델링 절차의 개요

객체지향 기반의 임베디드 소프트웨어 모델링 기법은 Gomaa의 COMET 방법론[18]이나 Douglass의 ROPES[19]와 Harmony 방법[20]들이 있다. 이 방법들은 모델 기반의 개발 접근 방법을 제시하고 있으나 하드웨어 아키텍처를 고려하는 플랫폼 의존적인 모델 부분에 대해서는 거의 언급하지 않고 있



(그림 1) 정의된 임베디드 소프트웨어 모델링 절차

다. 본 연구에서 제시한 모델링 절차는 멀티프로세서용 임베디드 소프트웨어의 모델링을 위한 방법으로 (그림 1)에서와 같이 크게 5단계를 거쳐 이루어진다[21]. 요구사항의 수집으로부터 PIM(Platform Independent Model) 모델링과 PSM (Platform Specific Model) 모델링 단계를 거쳐, 모델에 대한 검증(Verification & Validation) 과정을 수행하게 되며, 최종적으로 C/C++의 소스 코드를 생성하게 된다.

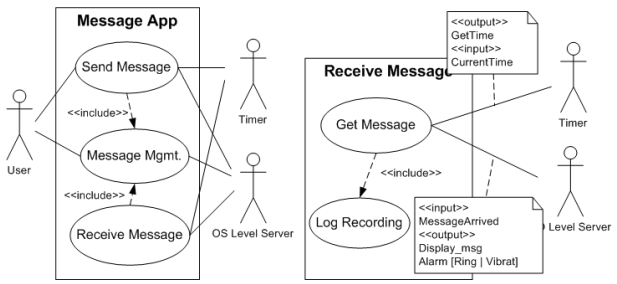
(그림 1)에서 보는바와 같이 임베디드 시스템의 소프트웨어 요구사항을 중심으로 UML 모델을 개발하며, 하드웨어 요구사항을 기반으로 멀티프로세서 하드웨어 아키텍처를 모델링 한다. 소프트웨어 모델과 하드웨어 아키텍처 모델은 상호 매핑되며, 이들은 분석 및 시뮬레이션 과정을 거쳐 요구사항 만족 여부를 검증하게 된다. 이와 같은 과정 중에서 소프트웨어 모델과 하드웨어 아키텍처 모델의 매핑은 다수의 프로세서로 태스크를 할당하기 위해 수행하는 과정으로써, 먼저 소프트웨어 모델을 CBCFG(Constraints-Based Control Flow Graph)로 변환하고, 이를 슬라이싱하여 최종 태스크를 도출하게 된다. 본 논문에서는 소프트웨어의 병렬성 향상과 하드웨어 자원의 효율적인 사용을 목적으로 UML 기반의 소프트웨어 모델로부터 태스크를 도출하는 방법을 제시한다.

3.2 소프트웨어 모델링

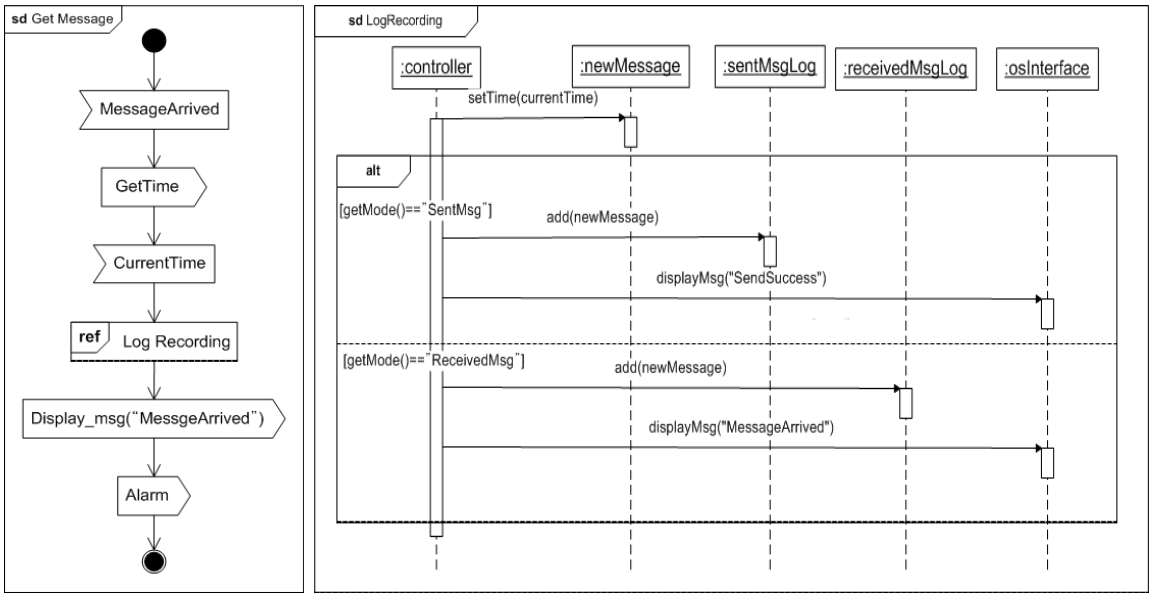
(그림 1)에서 정의된 PIM 모델링 단계에서는 임베디드 소프트웨어의 요구사항을 기반으로 UML 모델을 개발한다. 먼저 요구사항을 기반으로 계층 구조를 갖는 UCD(Use Case Diagram)을 작성하며, 이때 각 Use Case 단위로 입출력 이벤트들이 식별되고 Use Case 시나리오가 작성된다. 작성한 시나리오를 근거로 하나의 Use Case는 하나의 IOD로 변환하는데, 이때 IOD는 Use Case가 갖는 제어의 흐름을 표현하게 된다. IOD에 나타나는 각각의 'Activity'와 'Interaction Fragment'는 SD로 모델링되어 보다 상세한 행위 모델링을 표현하게 된다. 각 Use Case로부터 작성된 IOD들은 시스템의 전체 행위를 포함하기 위하여 하나의 IOD로 통합된다. IOD의 통합은 최상위 수준에서 모델링된 UCD를 Activity Diagram으로 변환함으로써 쉽게 이루어질 수 있다.

위와 같은 절차에 따라 UML 2.0 기반으로 모델링된 UCD, IOD, 그리고 SD는 (그림 2)와 같다. (그림 2)는 개인용 무선 단말기(휴대폰)의 메시징(messaging) 서비스에 대한 최상위 수준의 UCD와 이로부터 상세화된 "Receive Message"에 대한 UCD이며, (그림 3)은 상세화된 UCD로부터 생성된 IOD와 메시지 수신에 대한 로그를 기록하는 SD 모델이다.

(그림 3)의 왼쪽은 (그림 2)의 Use Case "Get Message"에 대한 IOD이며, 여기에 포함된 InteractionFagment "Log Recording"에 대한 SD가 오른쪽에 나타나 있다. 즉, 메시지가 도착하면 시스템으로부터 현재 시각을 알아내서 메시지 수신 정보에 대한 로그를 기록하는 동작이다.



(그림 2) 휴대 단말기에서의 MHS에 대한 UCD 모델



(그림 3) 휴대폰에서의 MHS에 대한 IOD와 SD 모델

(그림 3)의 IOD와 SD를 입력으로 CFG가 생성되는데, CFG는 IOD의 시작노드로부터 출발하여 모든 액티비티(activities)를 거쳐 최종 노드에 도달하는 과정에 대한 행위의 흐름으로 표현된다. 이 과정에서 IOD에 나타난 InteractionFagment는 해당 부분을 표현하는 SD에 의해 펼쳐져서 - Unfolding 되어 - 하나의 CFG로 표현된다.

4. 제약사항 기반 제어흐름그래프

4.1 CBCFG 정의

UML 2.0의 IOD와 SD를 입력으로 생성하는 제약사항 기반의 제어흐름 그래프(CBCFG)는 임베디드 소프트웨어가 갖는 행위 흐름을 표현하는 방향성 그래프로 다음과 같이 정의한다.

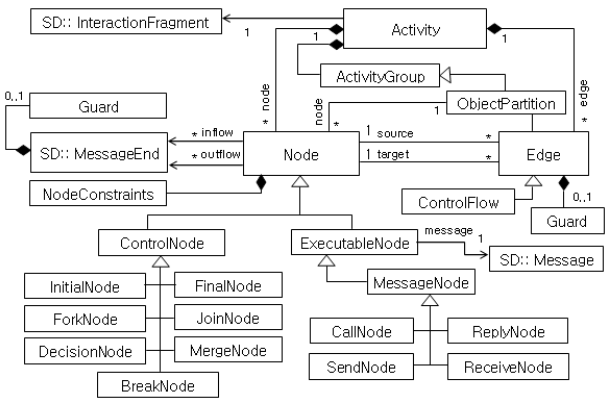
[정의 1] 제약사항기반의 제어흐름 그래프, CBCFG = (V, E, S, F, C)로 정의한다.
- V : 그래프에서의 노드(vertex)들의 집합
- E : 그래프에서 에지(edge)들의 집합, E ⊆ V X V
- S : 시작노드
- F : 끝 노드
- C : 노드 또는 에지가 갖는 제약사항(Constraint 또는 Guard)들의 집합

[정의 2] CBCFG의 구성요소 V는 다음과 같은 요소들로 정의한다.
- Vid : 노드의 유일한 식별자
- VT : 노드의 타입 ∈ {Send, Receive, Call, Reply, Fork, Join, Decision, Merge, Break}
- Vc ∈ C : 노드에 정의된 제약사항

[정의 3] CBCFG의 구성요소, C는 다음과 같은 문법에 의하여 정의한다.

Declaration of Node Signature;
NodeType: NodeName + Constraints
NodeType = {sn |rn |cn |rpn |fn |jn |dn |mn |bn };
NodeName = V_{id}; /* object name or message */
Declaration of Constraints;
Constraints = "("+{{Param}^{n} |Null}+")"+[From clause];
Param = src_object + dest_object;
if $V_T \in \{sn, rn, cn, rpn\}$
= iteration_condition; *if* $V_T \in \{dn\}$
From clause = "From" + ElementType ;
ElementType = {loop |par |alt | ... };

위와 같은 정의에 따라 표현된 CBCFG에 대한 메타 모델은 (그림 4)와 같다.



(그림 4) CBCFG의 메타모델

4.2 CBCFG 생성 규칙

4.1절의 정의에 따라 UML 기반의 소프트웨어 모델은 CBCFG로 변환된다. UML 2.0의 SD를 구성하고 있는 각 요소들에 대하여 CFG로 변환하는 방법에 대해서는 Garousi[9]에 의해 체계적으로 제안되었다. Garousi는 Rule-based 접근 방법을 통해 SD의 각 구성요소들에 대하여 CFG로 매핑하기 위한 14개의 규칙을 표 1과 같이 정의하였다.

<표 1>에서 정의된 규칙 R# 4와 #5의 매핑 규칙은 그림 5와 같다. (그림 5)는 SD의 비동기적 메시지 전달에 대하여 CFG로 매핑하는 형태를 보여준다. 비동기적인 메시지 전달의 경우에는 메시지 전달 자체를 CFG에서의 하나의 노드로 매핑하며, 비동기적 속성은 Fork 노드로 매핑되어 표현된다.

본 논문에서 제안하는 CBCFG를 생성하기 위하여 Garousi가 제안한 규칙 기반의 CFG 매핑 기법을 사용하였다. 그러나 기존에 제시된 규칙만으로는 CBCFG 생성이 어렵기 때문에 <표 2>와 같이 몇몇 규칙을 추가로 정의하였다. 규칙의 추가는 CBCFG 생성을 위해 IOD와 SD를 모두 입력으로 사용하기 때문이다.

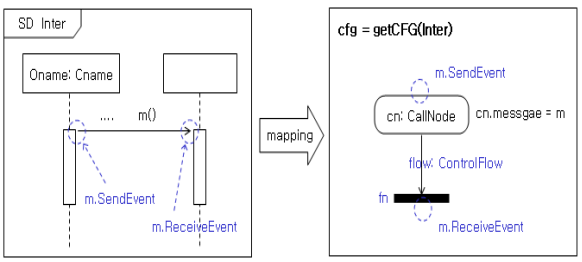
<표 2>의 규칙 R#15 ~ #18은 앞서 설명한 매핑 과정과 직관적으로 동일한 개념에 의해 진행한다. 그러나 R#19는 (그림 6)과 같은 과정을 거쳐 CBCFG로 매핑한다.

(그림 6)은 메시지 서비스에서 메시지 검색 과정에 'End' 버튼을 눌려 메시지 검색을 종료하는 인터럽트 처리(Interruptable Region)에 대한 IOD 부분의 매핑 규칙으로 InterruptableRegion은 CBCFG에서 BreakNode, bn으로 매핑된다.

<표 1>과 <표 2>에 주어진 매핑 규칙에 의거하여 생성된 (그림 3)의 메시징 서비스 기능에 대한 CBCFG는 (그림 7)과 같다.

〈표 1〉 SD의 구성요소에 대한 CFG 매핑 규칙 [9]

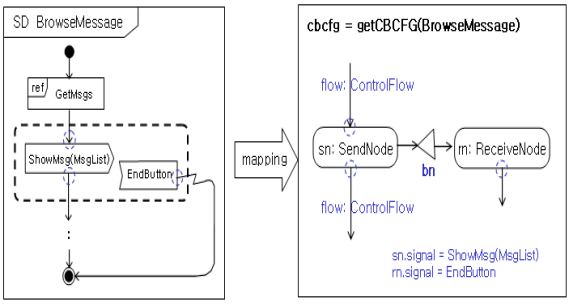
R#	SD Feature	CFG Feature
1	InteractionFragment	CFG
2	First message end	Flow between InitialNode and first control node
3	SynchCall/SynchSignal	CallNode
4	AsynchCall/AsynchSignal	(CallNode+ForkNode) or ReplyNode
5	Message "SendEvent" & Message "ReceiveEvent"	ControlFlow
6	LifeLine	ObjectPartition
7	par CombinedFragment	ForkNode
8	loop CombinedFragment	DecisionNode
9	alt/opt CombinedFragment	DecisionNode
10	break CombinedFragment	Edge
11	Last message ends	Flows between ending control nodes and FinalNode
12	InteractionOccurrence	Control Flow across CFGs
13	Polymorphic message	DecisionNode
14	Nested InteractionFeagment	Nested CFG



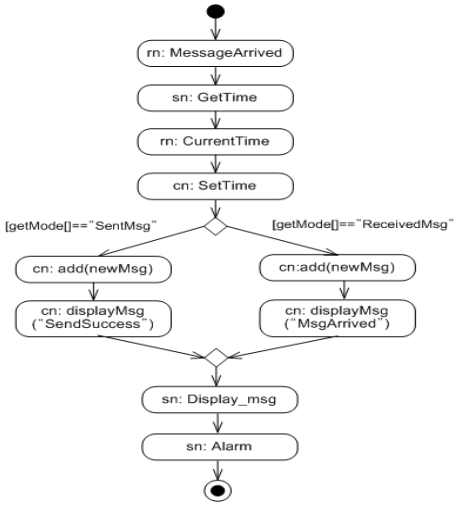
(그림 5) 규칙 R#4, 5에 대한 CFG 매핑 예제

〈표 2〉 CBCFG 생성을 위한 매핑 규칙

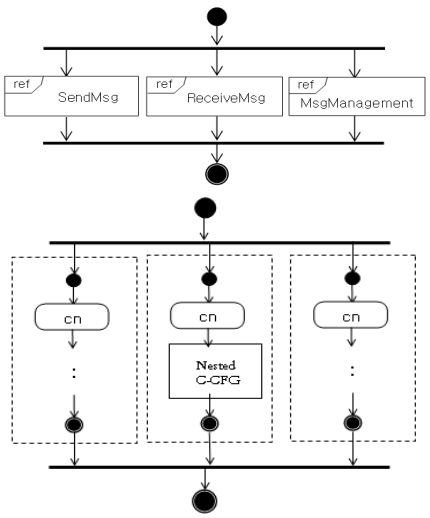
R#	IOD Feature	CBCFG Feature
15	SendSignal	SendNode
16	ReceiveSignal	ReceiveNode
17	ControlFlow	ControlFlow
18	ActivityNode	SD.InteractionFragment
19	InterruptableRegion	BreakNode
20	Fork/Join/Decision/Merge	ForkNode/JoinNode/ DecisionNode/MergeNode



(그림 6) 규칙 R#19의 CBCFG 매핑 규칙



(그림 7) MHS 모델(그림2)에 대한 단순화된 CBCFG



(그림 8) 최상위 수준 IOD(위)와 통합된 CBCFG(아래)

위와 같이 모든 IOD에 대하여 CBCFG가 생성되면 이들을 통합하기 위한 과정이 수행되는데, 이는 최상위 수준의 UCD를 기반으로 작성된 IOD의 흐름을 기반으로 통합이 이루어진다. (그림 8)은 생성된 CBCFG의 통합된 형태를 나타낸 것이다. 즉 최상위 수준의 IOD에 포함된 각각의 InteractionFragment에 대하여 CBCFG를 이용하여 Unfolding 시키게 된다. 통합된 CBCFG는 시스템이 갖는 전체 행위에 대한 제어 구조를 나타내는 최상의 CBCFG로써, 추후 Root 태스크로써의 역할을 수행한다.

5. 소프트웨어 모델의 분할 기법

멀티프로세서 아키텍처를 기반으로 하는 임베디드 시스템에서 소프트웨어 모델의 분할은 병렬 처리 및 자원 활용을 높이기 위한 목적으로 단위 태스크를 분리해 내는 과정이라고 할 수 있다. 따라서 앞의 4장에서 설명한 소프트웨어 모델은 적절한 시스템의 특성에 맞도록 분할되어야 한다. 이러한 분할을 위해서 고려하는 소프트웨어 모델의 분할 기준은 다음과 같다.

5.1 노드 유형 VT = “Fork”인 경우

CBCFG에 포함된 그래프의 노드가 “Fork” 노드의 유형을 갖는 경우 그래프는 분할된다.

```
1:  if (VT = fn) {
2:      for (i = 1..j, j = (# of •(fn)) ) {
3:          Si <- i_th of •(fn);
4:          Fi <- jn of the Si;
5:      }
6:  Replace the fn with sn;
7:  sn.Parameter.dest_object = {Si, i =1..j};
8:  }
```

라인 2부터 4까지는 Fork 노드의 Out-going edge의 수(# of •(fn)) 만큼 별도의 CBCFG로 슬라이싱을 위한 처리문이며, 라인 6과 7은 Fork 노드를 Send 노드로 대체함으로써, 다른 프로세스에 할당된 Fork 프로세스의 시작을 알리기 위함이다.

5.2 VT = dn & ElementType = “loop”인 경우

일반적으로 반복처리의 시맨틱을 갖는 루프 구조에서 데이터의 의존성이 없는 경우 병렬처리가 가능하다[22, 23]. 특히 파이프라인 방식에 의한 데이터 처리나 컴파일러에 의한 병렬성 식별 분야에서는 루프 구조의 데이터 의존성에 대한 많은 연구가 진행되었다[22]. 본 연구에서도 이와 같이 상호 데이터 의존성이 없는 루프에 대하여 병렬처리를 위한 태스크를 식별한다.

```
1: if (VT = dn && ElementType = "loop") {
        // mn: merge node as a pair node of the dn.
2:      if ( !DataDep(dn, mn)) {
3:          for ( i = 1..j, j = (# of iterations)) {
4:              Si <- dn;
5:              Fi <- mn;
6:          }
7:      Replace the dn with sn;
8:      sn.Parameter.dest_objet = {Si, i = 1..j};
9:      }
10: }
```

라인 2 : DataDep(dn, mn)은 데이터 의존성을 점검하기 위한 함수이며, 결과가 참인 경우 dn.Parameter가 갖는 iteration_condition에서의 반복 횟수만큼 병렬처리 가능하도록 CBCFG를 생성한다. 이러한 분할 기법은 일반적으로 멀티미디어 스트림을 처리하기 위한 프로그램 구조에서 많이 나타난다.

5.3 하드웨어 아키텍처의 특성을 고려한 분할

MPSoC와 같은 멀티프로세서 아키텍처에서는 ARM 코어나, DSP(Digital Signal Processor), Device Controller 등과 같은 다양한 유형의 프로세서(PE, Processing Element)들이 존재할 수 있기 때문에 하드웨어 구성요소들의 특성과 개수에 따라 태스크가 적절히 분리되어야 한다[3]. 앞의 5.1과 5.2절에서 정의된 분할 기준에 대하여 하드웨어 특성을 반영한 분할 방법을 다음과 같은 재정의할 수 있다.

```
// 5.1절의 분할 방법에 대한 재정의
1: if (VT = fn) {
2:      for (i = 1..j, j = (# of CPs) ) {
3:          Si <- i_th of •(fn);
4.          Fi <- jn of the Si;
5:      }
6:  Replace the fn with sn;
7:  sn.Parameter.dest_object = {Si, i = 1..j};
8:  }
```

라인 2는 하드웨어 아키텍처를 구성하는 코어(Core) 프로세서의 개수(# of CPs)만큼 fn 노드를 분할하기 위함이다.

```
// 5.2절의 분할 방법에 대한 재정의
1: if (VT = dn && ElementType = "loop") {
        // mn: merge node as a pair node of the dn.
2:      if ( !DataDep(dn, mn)) {
3:          p = 1;
4:          q = ⌈ (# of iterations) / (# of DSPs)⌉
5:          for ( i = 1..j, j = (# of DSPs)) {
6:              Si <- dn;
7:              Fi <- mn;
8:              Si -> dn.iteration_condition = (p .. (q*i));
9:              p = q * i + 1;
10:         }
11:         Replace the dn with sn;
12:         sn.Parameter.dest_objet = {Si, i = 1..j};
13:     }
14: }
```

즉, 라인 3, 4, 그리고 9는 병렬처리 가능한 for 루프의 반복 횟수를 DSP의 수에 맞도록 분할하기 위하여 반복 조건문을 변경하는 부분이다.

6. 예제 시스템 적용 및 분석

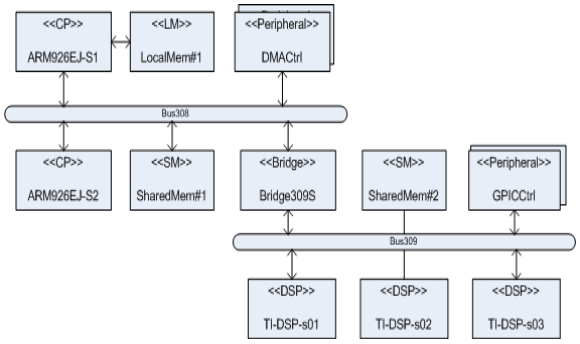
6.1 예제 시스템 명세

앞서 제시한 소프트웨어 모델에 대한 CBCFG 생성 및 분할 기법의 적용을 위한 예제 시스템은 휴대형 단말기에 내장되는 응용 소프트웨어들로써, 심비안(Symbian) OS[24]에 의해 지원되는 Messaging 서비스와 Multimedia 서비스 기능이다. 이들 기능에 대한 상위 수준의 요구사항을 정의하면, <표 3>과 같다.

위와 같은 소프트웨어의 요구사항을 구현할 하드웨어 아키텍처의 구성은 (그림 9)와 같다. (그림 8)의 하드웨어 구성은 ARM Core 2개, DSP 3개, 그리고 1개의 지역 메모리와 2개의 공유메모리, 주변장치 제어기 등으로 구성되어 있다.

〈표 3〉 메시징과 멀티미디어 서비스 요구사항

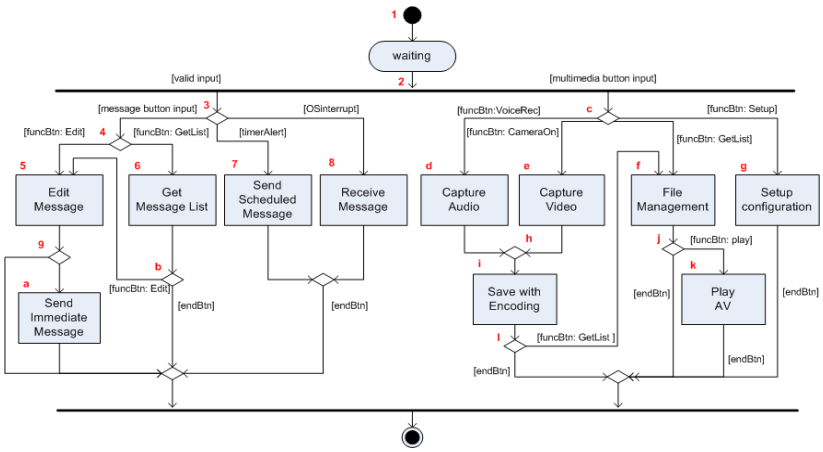
[메시징 서비스] s.1. 메시지 송신기능 s.1.1. 단순문자메시지 전송 s.1.2. 멀티미디어 메시지 전송 s.1.3. 최대 20명까지 동시 송신 가능 s.1.4. 예약 메시지 전송 기능 s.2. 메시지 수신기능 s.2.1. 메시지 수신에 대한 알림 기능 s.3. 메시지관리기능 s.3.1. 메시지 편집 및 저장 기능 s.3.2. 메시지 브라우징 기능 s.3.3. 메시지 송수신에 대한 로그 기록 s.3.4. 메시지 삭제 기능
[멀티미디어 서비스] m.1. 파일 생성기능 m.1.1. 음성 녹음 기능 m.1.2. 사진 촬영 기능 m.1.3. 동영상 촬영 기능 m.1.4. 파일 저장 기능 m.2. 파일 관리 기능 m.2.1. 파일 브라우징 기능 m.2.2. 파일 삭제기능 m.2.3 저장 파일 재생 기능 m.3. 환경설정기능 m.3.1. 파일 생성을 위한 환경 설정 기능



(그림 9) 예제시스템의 하드웨어 아키텍처 구성도

6.2 CBCFG 생성 및 분할 결과

앞의 6.1절에서 정의한 예제 시스템의 요구사항으로부터 UML 기반의 모델링을 통해 얻은 최상위 수준의 IOD 다이어그램은 (그림 10)와 같다.



(그림 10) 예제시스템의 최상위 수준 IOD 다이어그램

(그림 10)의 IOD 다이어그램은 초기 'waiting' 단계에서 메시징 서비스와 멀티미디어 서비스로 Fork 되는 형태를 표현하며, 'Edit Message'와 'Get Message List'는 메시지 기능 버튼의 입력에 의해 동작하며, 'Send Scheduled Message'와 'Receive Message'는 타이머와 OS로부터의 인터럽트에 의해 구동된다. (그림 10)의 오른쪽 부분은 멀티미디어 서비스 기능을 모델링 한 것으로써, 'Capture Audio', 'Capture Video', 'File Management', 'Setup' 등의 동작을 갖는다. (그림 10)에 나타난 각각의 Activity는 하위의 IOD로 모델링되거나 SD로 상세화 된다.

(그림 10)의 IOD 다이어그램을 입력으로 생성된 CBCFG의 목록은 <표 4>와 같다. <표 4>에서 보는 바와 같이 전체 시스템은 총 11개의 그래프를 생성하였다. 이의 결과로부터 주목할 사항은 (그림 10)에서 시작점 위치경로가 cbcfg123459a인 CBCFG의 경우는 IOD 모델링 과정에서 분리되어 표현되었지만, IOD 'Edit Message'에 종속적인 'extend' 관계의 시스템 행위이기 때문에 cbcfg12345에 통합되어 그래프가 생성되었다는 것이다.

<표 4>에 주어진 11번째 그래프는 전체 시스템의 행위를 표현하기 위한 제어 구조를 나타내는 CBCFG로써, 초기 시스템 행위를 활성화시키는 역할을 담당한다. <표 4>에서 주어진 초기의 CBCFG 목록을 입력으로 5장에서 정의한 그래프 분할 기법을 적용한 결과는 <표 5>와 같다.

<표 5>의 결과에서 11개의 CBCFG는 19개의 CBCFG로 분할되었다. 이러한 분할은 예제로 주어진 하드웨어 아키텍처가 2개의 ARM 코어와 3개의 DSP로 구성된 특성을 반영하였기 때문이다. 특히, <표 5>에 나타난 cbcfg12345001과 cbcfg12345002는 cbcfg12345로부터 분할되었는데, 이것은 문자 메시지의 전송(요구사항 s.1.1) 행위가 CDMA 컨트롤러에게 비동기 메시지를 전송하는 방식으로 이루어지기 때문이다. (그림 11)은 분할 기법이 적용된 cbcfg12cdhi001 ~ cbcfg12cdhi004의 CBCFGs를 보여준다.

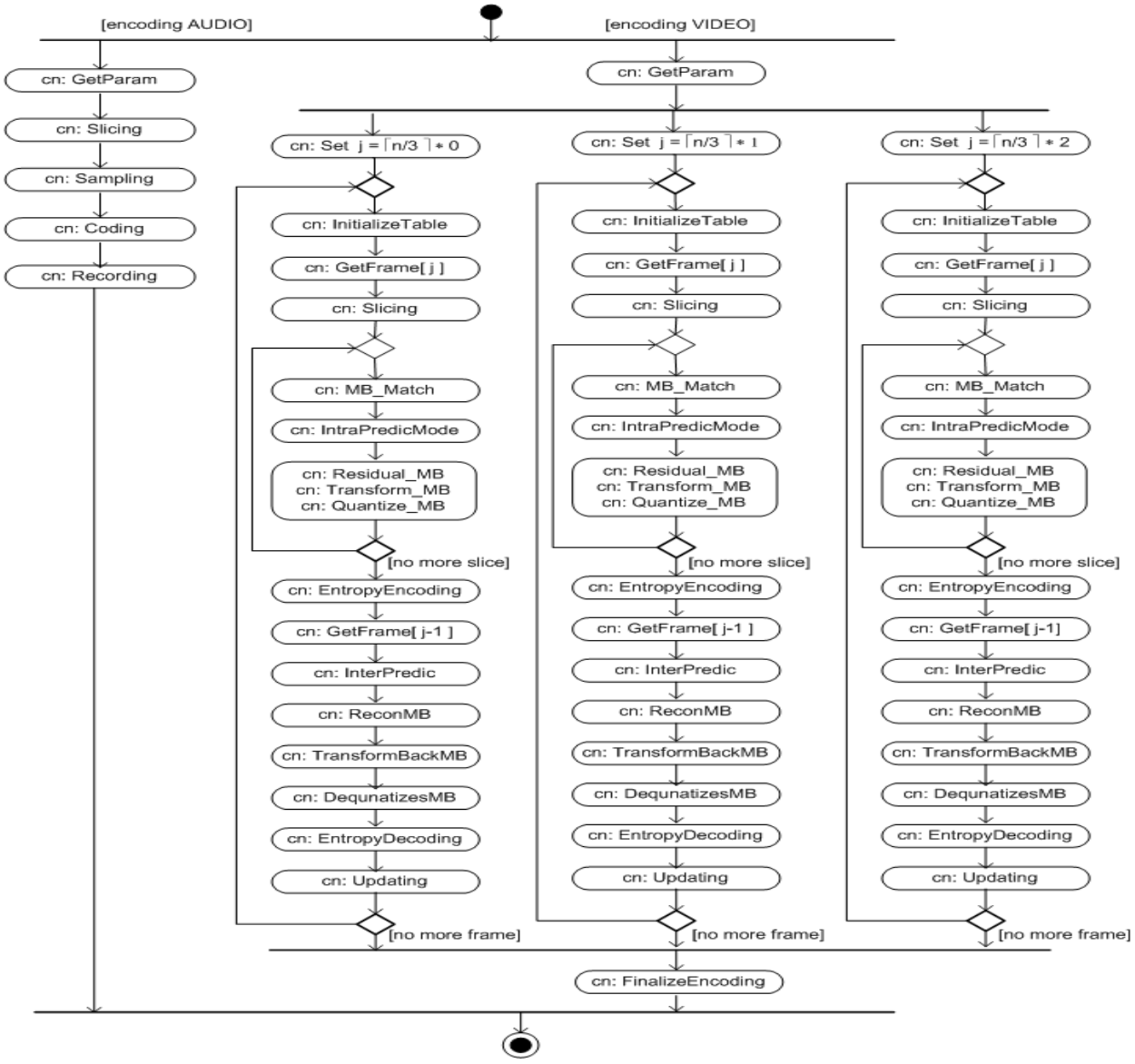
(그림 11)에서 보는 바와 같이 멀티미디어 서비스를 위한 인코딩 부분은 음성과 영상을 인코딩하는 부분으로 크게 구분하고, 영상 인코딩을 위해서는 입력되는 프레임을 등분하여 3개의 DSP에 각각 할당하도록 분할하였다. 각 DSP에서는 프레임내의 공간적 중복을 제거하는 Intra Transformation과 프

〈표 4〉 예제시스템의 초기 CBCFG 목록

번호	식별자	설명
1	cbcfg12345	edit message including send immediate message
2	cbcfg12346	get message list
3	cbcfg1237	send scheduled message
4	cbcfg1238	receive message
5	cbcfg12cd	capture audio input
6	cbcfg12ce	capture video input
7	cbcfg12cf	multimedia file management
8	cbcfg12cg	setup configuration
9	cbcfg12cdhi	save file with encoding
10	cbcfg12cfjk	play(decoding) audio and/or video
11	cbcfg1	control structure for whole behavior

〈표 5〉 분할된 CBCFG 목록

번호	식별자	분할 기준 설명
1	cbcfg12345001	메시지 송신과정에서 발생하는 비동기 호출에 의한 fork노드의 생성 및 분할
2	cbcfg12345002	
3	cbcfg12346	분할 없음
4	cbcfg1237	분할 없음
5	cbcfg1238	분할 없음
6	cbcfg12cd	분할 없음
7	cbcfg12ce	분할 없음
8	cbcfg12cf	분할 없음
9	cbcfg12cg	분할 없음
10	cbcfg12cdhi001	멀티미디어 파일을 인코딩하여 저장 하기 위한 제어 기능
11	cbcfg12cdhi002	멀티미디어 파일의 저장을 위한 코덱의 DSP 처리 부분
12	cbcfg12cdhi003	
13	cbcfg12cdhi004	
14	cbcfg12cfjk001	멀티미디어 파일을 재생하기 위한 제어 기능
15	cbcfg12cfjk002	멀티미디어 파일을 재생하기 위한 코덱의 DSP 처리부분
16	cbcfg12cfjk003	
17	cbcfg12cfjk004	
18	cbcfg1001	메시지 처리와 멀티미디어 처리에 대한 fork 노드의 분할
19	cbcfg1002	



(그림 11) 예제시스템의 인코딩 기능에 대한 분할된 CBCFGs

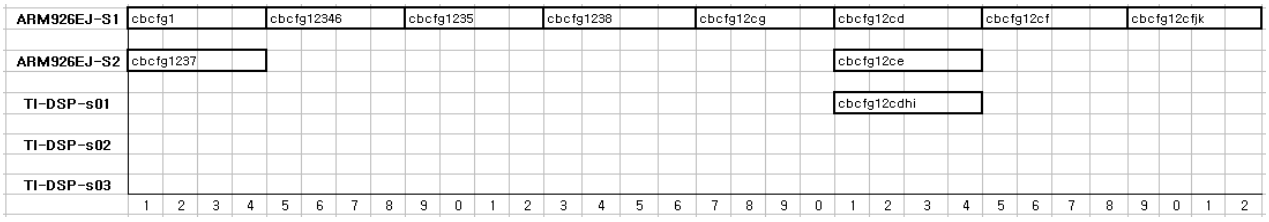
레임간의 시간적 중복을 제거하는 Inter Prediction 과정에 대하여 병렬처리를 수행하게 되며 이를 통해 보다 신속한 멀티미디어 서비스를 가능하게 한다.

6.3 분할 결과 분석

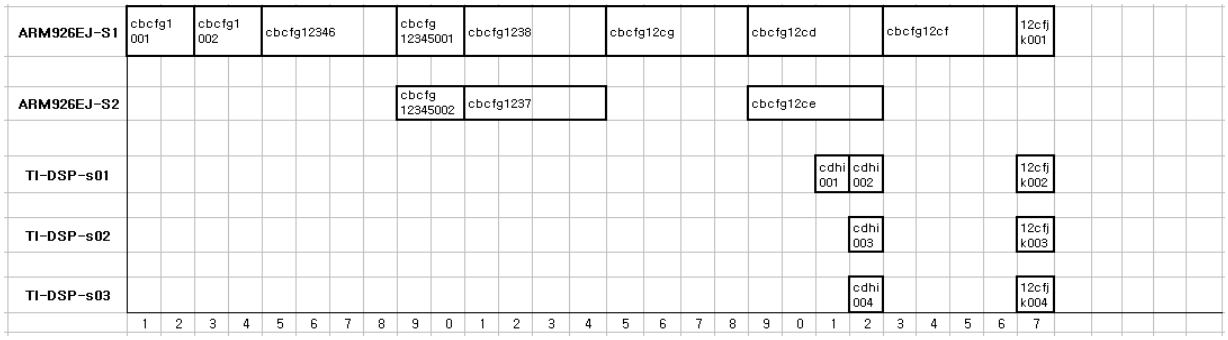
본 절에서는 분할 기법의 적용이 임베디드 시스템 개발에 어느 정도의 유용성을 제공하는 가에 대하여 살펴보았다. 특히 분할기법의 적용에 따른 병렬성의 향상 정도와 프로세서 자원의 활용성 측면에서 고찰하였으며, 다음과 같은 전제사항은 가정하였다.

- <표 4>에 정의된 11개의 각 CBCFG는 노드의 구현에 따라 수행 시간(즉, 자원 점유시간)이 다를 수 있다. 그러나 병렬성 및 자원 활용도 관점에서 분할 결과를 상대적으로 분석하는 것이기 때문에 모든 태스크는 동일한 1 단위의 자원 점유 시간을 갖는 것으로 가정하였다.
- CBCFG가 분할될 때, 자원의 점유 시간도 균등하게 분할된다.
- 6.1절에 정의된 예제 시스템의 요구기능을 기준으로 하되, 태스크의 실행 순서를 고려하기 위하여 다음과 같은 시나리오를 정의하였다.

> 사용자가 저장된 메시지 리스트로부터 메시지를 로딩하여 편집하고 전송한다. 이 과정에서 이미 저장된 예약 메시지가 전송되며, 새로운 메시지가 수신되었음을 알린다.
>
> 사용자는 동영상 촬영을 위해 해상도를 지정하고 동영상을 촬영한다. 촬영한 동영상을 저장하고, 저장된 동영상을 로딩하여 재생한다.



(그림 12) 분할전 CBCFGs의 할당 형태



(그림 13) 분할된 CBCFGs의 할당 형태

6.3.1 병렬성 분석

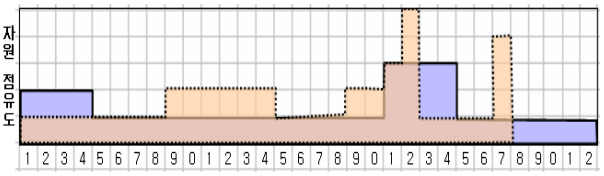
<표 4>와 <표 5>에 주어진 CBCFGs를 (그림 9)에 주어진 하드웨어 아키텍처에 할당하는 경우, 각각 어느 정도의 병렬성이 존재하는 가를 살펴보면 (그림 12), (그림 13)과 같이 표현할 수 있다. 이는 앞서 정의한 적용 시나리오의 실행 순서를 기준으로 CBCFG들의 전후 관계를 고려한 매핑 결과이다.

(그림 12)의 cbcfg1238은 메시지를 수신하는 경우로써, CDMA 프로토콜로부터 직접 인터럽트를 받아 수행되기 때문에 ARM926EJ-S1에 할당하였으며, (그림 13)에서 cbcfg1001와 cbcfg1002는 심비안 OS와의 인터페이스 부분을 담당하는 역할을 제공하기 때문에 ARM926EJ-S1에 할당하였다. (그림 12)와 (그림 13)로부터 <표 6>과 같은 병렬성의 정도를 산출할 수 있다.

<표 6>에서 최대 병렬성은 (그림 12)와 (그림 13)에서 시간축(X축)에서의 최대 동시 수행 태스크 수를 의미하고, 전체 병렬성은 모든 태스크에 대하여 병렬성을 더하고, 이를 전체 시간에 대한 평균값으로 산출한 것이다.

〈표 6〉 예제 시스템에 대한 병렬성의 정도

기준	분할 전	분할 후
최대 병렬성	3	5
전체 병렬성	1.35	1.63



(그림 14) 하드웨어 자원의 점유 정도

6.3.2 자원 활용성 분석

앞의 (그림 12)와 (그림 13)를 기준으로 하드웨어 자원의 점유 상태를 살펴보면 (그림 14)와 같이 표현할 수 있다. (그림 14)에서 점선으로 연결되는 부분은 분할 후의 자원 점유 정도이며, 실선으로 연결되는 부분은 분할 전의 자원 점유 정도를 표현한다.

(그림 14)로부터 분할 전과 분할 후의 자원 점유정도를 산출하면 <표 7>과 같다. 점유도 값은 전체 사각형의 면적 대비 자원의 점유 면적을 산출한 값이다. 즉 분할 전의 자원 점유도 값은 44/160의 값으로 계산되며, 분할 후의 점유도는 44/135로 정의할 수 있다.

위와 같은 결과로부터 본 연구에서 제시하는 소프트웨어 모델의 분할 기법을 적용하면, 분할하지 않고 하드웨어에 할당하는 것에 비해 병렬 처리 측면에서는 22%, 자원 활용도 측면에서는 18.5%의 향상이 있음을 알 수 있다.

7. 결론 및 향후 연구내용

본 논문은 UML 2.0 다이어그램을 이용하여 표현된 임베디드 소프트웨어 모델을 멀티프로세스 아키텍처로 매핑하기 위해 모델 분할 기법을 제시하였다. 이를 위하여 UML의 IOD와 SD를 입력으로 CBCFG를 생성하고, 생성된 CBCFG

〈표 7〉 예제 시스템에 대한 자원 점유도

	분할 전	분할 후
자원 점유도	27.5%	32.6%

를 하드웨어 구성요소들의 특성을 고려하여 분할하는 과정 및 기법에 대하여 정의 하였다. 제시한 기법의 적용은 임베디드 소프트웨어의 병렬성을 향상시키고, 또한 하드웨어 자원의 활용성을 높일 수 있다는 정점을 제공할 수 있다.

본 연구는 임베디드 소프트웨어의 복잡성 증가로 인하여 수요 증대가 예측되는 MPSoC와 같은 멀티프로세서 아키텍처를 위한 태스크 분할 및 할당 기법에 적용하기 위한 것으로써, 향후 그 소요 및 활용이 매우 높을 것으로 예측되는 분야이다. 특히 멀티미디어 응용이나 암호화 알고리즘 등의 영역에서 매우 유용할 것으로 판단한다.

본 논문에 이어 추가적으로 수행하고자 하는 연구는 최근 임베디드 시스템 개발에서 많은 관심을 갖는 에너지 소비량을 줄이는 것으로써, 본 연구에서 제시하는 소프트웨어 모델의 분할과정에서 에너지의 효율성을 고려한 모델 분할 기법을 제안하는 것이다.

참 고 문 헌

[1] B. Horowitz, et. al., "Platform-Based Embedded Software Design and System Integration for Autonomous Vehicles," Proceedings of the IEEE, Vol.19, No.1, pp.198-211, Jan. 2003

[2] 김우열, 김영철, "확장된 xUML을 사용한 MDA 기반 이종 임베디드 소프트웨어의 컴포넌트 모델링에 관한 연구, 정보처리학회논문지D, Vol.14-D, No.1, pp.83-88, 2007년 2월.

[3] A. Jerraya, and W. Wolf, 'Multiprocessor Systems-on-Chip', Morgan Kaufmann, Sep., 2004.

[4] Giovanni De Micheli, "MPSoc HW Challenges - Reliability and Reliable Design," Tutorial of MPSoC'05, France, July, 2005.

[5] OMG, 'UML 2.0 Superstructure Specification', Doc #ptc-04-10-20, Oct., 2004.

[6] W. Plishker, et. al., "Automated Task Allocation on Single Chip, Hardware Multithreaded Multiprocessor Systems," Workshop on Embedded Parallel Architecture(WEPA-1), pp.14-20, Feb., 2004.

[7] T.S. Dahl, et. al, "Scheduling with Group Dynamics: a Multiple-Robot Task Allocation Algorithm based on Vacancy Chains," Technical Report CRES-002-07, Center for Robotics and Embedded Systems, Univ. of Southern California, 2002.

[8] P.G. Paulin, et. al., "Parallel Programming Models for Multi-Processor SoC Platform Applied to High-Speed Traffic Management," The Proceedings of CODES'04, pp.48-53, Sep., 2004.

[9] V. Garousi, L. Briand, and Y. Labiche, 'Control Flow Analysis of UML 2.0 Sequence Diagrams,' Carleton University TR SCE-05-09, Sept., 2005.

[10] H. Storrle, "Semantics of Control-Flow in UML 2.0 Activities," Proceedings of IEEE VLHCC'04, pp.235-242, Sept., 2004.

[11] P. Blasio, K. Fisher, and C. Talcott, "A Control-Flow Analysis for a Calculus of Concurrent Object," IEEE TSE, Vol.26, No.7, pp.617-634, July, 2000.

[12] J. Zhao, "Control-Flow Analysis and Representation for Aspect-Oriented Programs," Proceedings of QSIC'06, Beijing Chian, pp.38-48, Oct., 2006.

[13] H. Hollstein, et. al., "HiPART: A New Hierarchical Semi-Interactive HW/SW Partitioning Approach with Fast Debugging for Real-Time Embedded Systems," Proceedings of the CODES'98, pp.29-33, USA, Mar, 1998.

[14] G. Vanmeerbeek, et. al., "Hardware/Software Partitioning of Embedded System in OCAPI-xl, Proceedingd of CODES'01, Denmark, pp.30-35, 2001.

[15] H. Cornelis and E.D. Schutter, "Neurospaces: Towards Automated Model Partitioning for Parallel Computers," Journal of Neurocomputing, Vol.70, pp.2117-2121, 2007.

[16] K. Hering, G.Runger, and S. Trautmann, "Modular Construction of Model Partitioning Process for Parallel Logic Simulation," Parallel Processing Workshop, 2001, pp.99-105, Sept. 2001.

[17] N. Li, Y-J. Fang, "Software/hardware Patition in Multiple Processors Embedded System," Proceedings of 4th ICMLC'05, pp.165-170, Aug., 2005.

[18] H. Gomaa, 'Designing Concurrent, Distributedm and Real-Time Applications with UML', Addison-Wesley, 2000.

[19] B. P. Douglass, 'Real Time UML', 3rd ed., Addison-Wesley, 2004.

[20] B. P. Douglass, 'Real-Time UML Workshop for Embedded Systems', Newnes, 2007

[21] Jang-Eui Hong and Doo-Hwan Bae, "MDA-Based Embedded Software Development for Multiprocessor Architecture," Communications of KISS, Vol.24, No.8, pp.19-25, 2006.

[22] K. Kyriakopoulos and K. Psarris, "Data Dependence Analysis for Complex Loop Regions," pp.195-204, Proceedings of Parallel Processing, pp.195-204, Sept., 2001.

[23] T. Jacobson and G. Stubbendieck, "Dependency Analysis for For-Loop Structures for Automatic Parallelization of C Code," Proceedings of the MICS 2003, Duluth, April, 2003.

[24] Symbian, 'Symbian OS Ver 9.1 Product description', Revision 1.1, Feb., 2005.

김 종 필

e-mail : kimjp@selab.chnugbuk.ac.kr

2006년 충북대학교 컴퓨터공학과(학사)

2006년~현 재 충북대학교 전자계산학과 석사과정

관심분야 : 임베디드 소프트웨어, 소프트웨어 아키텍처, 소프트웨어 품질공학

홍 장 의

e-mail : jehong@chungbuk.ac.kr

1988년 충북대학교 전자계산학과(학사)

1990년 중앙대학교
전자계산학과(공학석사)

2001년 한국과학기술원 전산학(공학박사)

2002년 국방과학연구소 선임연구원

2002년~2004년 (주)솔루션링크 기술연구소장

2004년~현 재 충북대학교 컴퓨터공학 조교수

관심분야 : 소프트웨어공학, 임베디드 소프트웨어, 소프트웨어
품질공학, 소프트웨어 프로세스